Security in Web and Database applications

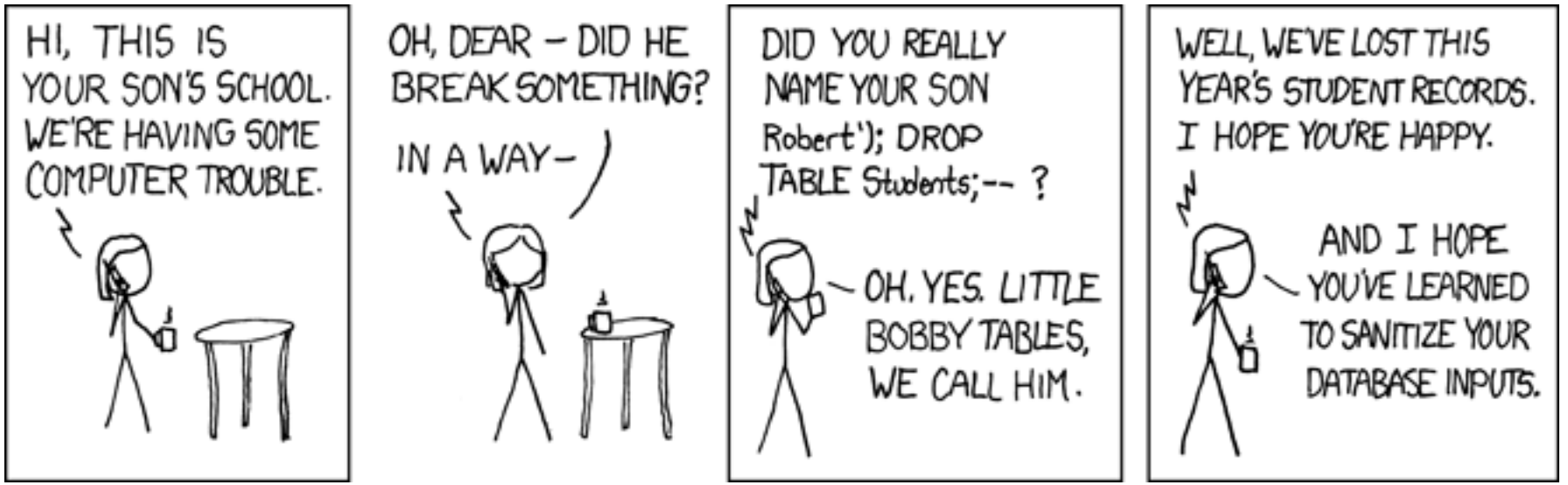
CS-370: Software Design and Development

Robert Bruce

Tips for building secure web and database applications

- Sanitize all inputs to avoid SQL and HTML injection attacks.
- Use strong encryption algorithms in your applications.
  - I suggest the Advanced Encryption Standard (AES) with a 256-bit key.
- Include a built-in time-out function which automatically logs the user out of their account after several minutes of inactivity.
  - Delete all current sessions in the database (server-side) to indicate the user must re-authenticate to continue any web-based transactions requiring login.
- Use some sort of session management when users log in to maintain state (i.e. remember the user).
  - Minimally encrypt the session data on the server-side to prevent client-side from tampering with the value.
- Keep a data log for all user login transactions.
  - Store the client's IPv4 or IPv6 internet addresses in a database whenever the client logs in.
  - Store the date and time when the client has logged in.
  - Possibly store the date and time when the client has logged out (or timed-out).
- Encrypt all your database tables.

Obligatory XKCD comic about SQL injections



Source:
https://imgs.xkcd.com/comics/exploits_of_a_mom.png

Example: SQL injection attack

- Build an SQL query dynamically using unsanitized input string values for *first_name* and *last_name*:
  - ```
    snprintf (&db_statement[0], MAX_DB_STATEMENT_SIZE-1, "INSERT INTO students (first_name, last_name) VALUES ('%s', '%s')", &first_name[0], &last_name[0]);
    ```
- Let *first_name* be the following literal string (in blue):
  - ```
    Ha', 'Ha'); DELETE FROM students; INSERT INTO students (first_name, last_name) VALUES ('Your data
    ```
- Let *last_name* be the following literal string (in green):
  - ```
    is gone
    ```
- When these variables are substituted into our snprintf() statement, the resulting SQL query is as follows:
  - ```
    INSERT INTO students (first_name, last_name) VALUES ('Ha', 'Ha'); DELETE FROM students; INSERT INTO students (first_name, last_name) VALUES ('Your data', 'is gone');
    ```
- When executed by MySQL, the statement above will actually execute three commands:
  - ```
    INSERT INTO students (first_name, last_name) VALUES ('Ha', 'Ha');
    ```
  - ```
    DELETE FROM students;
    ```
  - ```
    INSERT INTO students (first_name, last_name) VALUES ('Your data', 'is gone');
    ```
- *This is an SQL injection attack with really nasty consequences!*
    ```

How to avoid SQL injections?

- When building SQL queries, sanitize all of your inputs by safely escaping special MySQL reserved characters.
- How do you sanitize inputs? Use the backslash to escape the following special characters in MySQL queries:
  - A null character (not to be confused with the NULL reserved word): `\0`
  - A single quote (') character: `\'`
  - A double quote (") character: `\"`
  - A backspace character: `\b`
  - A newline (linefeed) character: `\n`
  - A carriage return character: `\r`
  - A tab character: `\t`
  - A backslash "\" character: `\\`
  - A percent "%" character: `\%`
  - An underscore "_" character: `\_`

How to avoid HTML injections?

- When displaying output to the user's browser, it must be formatted in HTML.
- It is important that any dynamic data placed within a web page (to be displayed on the user's browser) does not contain special HTML character entities that could be interpreted as HTML on the user's browser.
- How do you avoid HTML injections? At a minimum, the following characters should be converted to their equivalent character entities:
  - A space character: ` `
  - A double quote (") character: `&quot;`
  - An ampersand (&) character: `&`
  - A slash (/) character: `&#47;`
  - A less-than (<) character: `<`
  - A greater-than (>) character: `>`

Example: Avoiding HTML injection

- Example: a user has created a new user account via your auction application with the username: **BOLD**
- In this example, the user's name is literally "**BOLD**". Anytime we display that user's name on a web page it will be displayed as **BOLD** text since the **** and **** in the username will be interpreted by the client's web browser as HTML tags.
- To alleviate this, we must santize the username by replacing certain characters with their equivalent HTML character entities (like an ASCII code). In this case, the username would be encoded as: `<b>BOLD<&#47;b>` and displayed on the user's browser (literally) as **BOLD**
- *Note: while the HTML injection example (above) is simple and relatively harmless, do not underestimate XSS (cross-site scripting) as another technique in HTML injection. For more information about XSS attacks, see https://brightsec.com/blog/xss-attack/*

Encryption using the GNU crypto library

- The GNU crypto library provides many different encryption algorithms.
- My personal preference is the Advanced Encryption Standard (AES) with a 256-bit key.
- AES is a symmetric-key cryptographic algorithm. This means, the same key is used to encode and decode a message.
- *Note: I have provided an example C program on Canvas to encrypt and decrypt a message using AES 256 bit keys. Please see Canvas files folder → Example programs → encryption example.*

Creating a session: step-by-step

- Let's define some terminology to make it easier to represent the flow of information during an HTTPS session:
  - **Client**: the IP (internet-protocol) address of the user who is accessing the web server via a web browser.
  - **Blue web server**: the web-server running on a computer named Blue in the Computer Science department at Sonoma State Univeristy. The *Blue web server* is responsible for delivering all web-page content to a *Client* (web browser).
  - **Auction app**: the auction application that runs on the Blue computer server and receives all input from the *Client* (web browser) via the Common Gateway Interface (CGI) on the *Blue web server*. The application also formats all output in HTML which is delivered to the *Client* (web browser) via the *Blue web server* as well.
  - **MySQL server**: the database server running on a computer named Blue in the Computer Science department at Sonoma State University. The MySQL server accepts SQL (structured query language) commands and sends results (if applicable) from your Auction app running on the Blue computer server.

Stage 1: Creating a session

1 The *Client* requests “login.cgi” from the *Blue web server.*

2 The *Blue web server* executes the *Auction app*.

3 The *Auction app* outputs HTML login web page to the *Blue web server.*

4 The *Blue web server* displays HTML login web page to the *Client*.

Stage 2: Creating a session

1 The *Client* enters “O’Flaherty” username in the username field and “super%secret” in the password field of the HTML web login form.

2 The *Client* presses the “submit” button.

3 The *Client* submits following variable-value pairs to Blue web server via POST protocol: username=O%27Flaherty&password=super%25secret

4 The *Blue web server* sends the following variable-value pairs to the *Auction app* via the Common Gateway Interface using the POST protocol: username=O%27Flaherty&password=super%25secret

5 The *Auction app* separates the variable-value pairs (note: “%27” is translated into a single quote mark while “%25” is translated into a percent symbol):

username

O’Flaherty

password

super%secret

Stage 2: Creating a session (continued)

6 The *Auction app* creates a safely encoded version of user-input username “O’Flaherty” in a variable named safe_sql_username. The contents of safe_sql_username is: `O\'Flaherty`

7 The *Auction app* creates a safely encoded version of user-input username “super %secret” in a variable named safe_sql_password. The contents of safe_sql_password is: `super\%secret`

8 The *Auction app* builds an SQL query: `SELECT COUNT(*) FROM users WHERE username="$safe_sql_username" AND password="$safe_sql_password"`

9 The *Auction app* sends the query (from step 8) to *MySQL* with values substituted for each variable: `SELECT COUNT(*) FROM users WHERE username="O\'Flaherty" AND password="super\%secret"`

Stage 2: Creating a session (continued)

10 The *MySQL server* returns "1" for our last query. This count indicates the user correctly authenticated in our system since their username matches a username registered in our system and their password also matches. *Note: the count should NEVER be more than one since username is primary key in the username table*.

11 The *Auction app* retrieve the environment variable "REMOTE_ADDR" from the *Blue web server*. This variable stores the IPv4 or IPv6 address for the *Client*. Store this IP (internet protocol) address in a string variable called ip_address.

12 The *Auction app* sends a command to the *MySQL server* to execute: `SELECT now();`

13 The *Auction app* stores the results of the query from step 11 into a variable called timestamp. Timestamp will be formatted as "YYYY-MM-DD HH:MM:SS". Example: "`2025-10-13 10:32:13`".

14 The *Auction app* creates a new variable called session which is an amalgamation of the variable ip_address, timestamp, and safe_username. Example: `107.115.29.22 2025-10-13 10:32:13 O'Flaherty`

15 The *Auction app* stores session as a series of two-digit ASCII hex bytes. The contents of session is: `31 30 37 2e 31 31 35 2e 32 39 2e 32 32 20 32 30 32 35 2d 31 30 2d 31 33 20 31 30 3a 33 32 3a 31 33 20 4f e2 80 99 46 6c 61 68 65 72 74 79`

16 The *Auction app* encrypt the contents of variable session with Advanced Encryption Standard and a 256-bit key. The AES key will ONLY be stored on the server. The encrypted contents of session (displayed as ASCII hex bytes) is then: `69 f7 55 d6 5f 0a 59 65 74 57 c8 46 f5 78 1a c8 fc 79 88 70 8a 2c de c1 ac e7 b1 1c 9a 1a cd ca f2 11 6b 66 a9 c9 00 0e 9f ce 08 e6 60 c9`

17 The *Auction app* packs the contents of session by removing spaces between two-digit hex values and storing in a variable named packed_session: `69f755d65f0a59657457c846f5781ac8fc7988708a2cdec1ace7b11c9a1acdcaf2116b66a9c9000e9fce08e660c9`

Stage 2: Creating a session (continued)

18 The *Auction app* creates an SQL query to store the contents of packed_session in the *MySQL server*: `UPDATE user SET session = "69f755d65f0a59657457c846f5781ac8fc7988708a2cdec1ace7b11c9a1acdcaf2116b66a9c9000e9fce08e660c9" WHERE username = "O\'Flaherty";`

19 The *Auction app* sends the SQL query from step 18 to the *MySQL server*.

20 The *Auction app* displays some logged-in only content to the user. The *Auction app* should encode each URL inside the members-only content (i.e. requires user login) with the following:
"`session=69f755d65f0a59657457c846f5781ac8fc7988708a2cdec1ace7b11c9a1acdcaf2116b66a9c9000e9fce08e660c9`"

Stage 3: Authenticating with session

1 Scenario: user already logged in to the *Auction app*. Now we pass-in a session variable-value pair to remember our user. Now check if a variable-value pair exists for a variable called "session" (sent via Common Gateway Interface via GET protocol). Example: `session=69f755d65f0a59657457c846f5781ac8fc7988708a2cdec1ace7b11c9a1acdcaf2116b66a9c9000e9fce08e660c9`

2 If you receive a session variable-value pair in *Auction app* do the following: First separate the variable value pair:

- session
- 69f755d65f0a59657457c846f5781ac8fc7988708a2cdec1ace7b11c9a1acdcaf2116b66a9c9000e9fce08e660c9

3 In the *Auction app*, decrypt the AES encrypted value with the same key you encrypted it with store it in the variable decrypted_session. Example: "`107.115.29.22 2025-10-13 10:32:13 O'Flaherty`".

4 In the *Auction app*, extract the IP address from decrypted_session and store that value in session_IP_address. Example: "`107.115.29.22`".

5 In the *Auction app*, extract the date-timestamp value from decrypted_session and store that value in session_date_timestamp. Example: "`2025-10-13 10:32:13`".

Stage 3: Authenticating with session (continued)

6 In the *Auction app*, prepare another SQL query: `SELECT TIMESTAMPDIFF (MINUTE, "session_date_timestamp", NOW());`

7 Execute the query (from step 7) in *MySQL server*: `SELECT TIMESTAMPDIFF (MINUTE, "2025-10-13 10:32:13", NOW());`

8 In step 7, *MySQL server* will return the number of minutes that have elapsed since your date-timestamp and current time. If the amount of time meets or exceeds three minutes, prompt the user to log-in again. If the time is less than three minutes, proceed with step 9.

9 In the *Auction app*, retrieve the environment variable “REMOTE_ADDR” from the *Blue web server*. This variable stores the IPv4 or IPv6 address for the *Client*. Store this IP (internet protocol) address in a string variable called ip_address.

10 In the *Auction app*, compare ip_address with session_IP_address. If these two values do not match, prompt the user to log-in again. If the two values match, proceed to step 11.

11 In the *Auction app*, prepare a safe version of session which will not cause a MySQL injection. Specifically, escape any special characters that would cause MySQL to fail. Store the result in safe_session.

12 In the *Auction app*, prepare another SQL query: `SELECT username FROM users WHERE session = “safe_session”;`

13 Execute the query (from step 11) in *MySQL server*: `SELECT username FROM users WHERE session = “69f755d65f0a59657457c846f5781ac8fc7988708a2cdec1ace7b11c9a1acdcaf2116b66a9c9000e9fce08e660c9”;`

14 In the *Auction app*, retrieve the username from MySQL server. If no username is returned, prompt the user to log-in again with their user-name password. If a username is returned, retrieve it. We now know who the user is and will proceed with displaying user-login-only web content.